Don't Fall into a Trap: Physical Side-Channel Analysis of ChaCha20-Poly1305

Bernhard Jungk, Shivam Bhasin
PACE, Temasek Laboratories
Nanyang Technological University, Singapore
{bjungk,sbhasin}@ntu.edu.sg

***Abstract*—The stream cipher ChaCha20 and the MAC function Poly1305 have been published as IETF RFC 7539. Since then, the industry is starting to use it more often. For example, it has been implemented by Google in their Chrome browser for TLS and also support has been added to OpenSSL, as well as OpenSSH. It is often claimed, that the algorithms are designed to be resistant to side-channel attacks. However, this is only true, if the only observable side-channel is the timing behavior. In this paper, we show that ChaCha20 is susceptible to power and EM side-channel analysis, which also translates to an attack on Poly1305, if used together with ChaCha20 for key generation. As a first countermeasure, we analyze the effectiveness of randomly shuffling the operations of the ChaCha round function.**

I. Introduction

The stream cipher ChaCha20 and the authentication code Poly1305 are increasingly implemented in several important and widespread products, such as Google Chrome [1], or OpenSSL [2]. Based on these prominent examples, the two algorithms have potential to be adopted across multiple domains in the future. The ChaCha20-Poly1305 cipher suite is advertised as being easier to implement in a side-channel resistant way [3], especially compared to ciphers based on substitution permutation networks. For example, many implementations of AES are susceptible to timing side-channel attacks [4].

However, it is possible to extract the key material using other side-channels, such as the power [5] or the EM side-channel [6], because ChaCha20 has no built-in protection against these attacks. For classical network infrastructures such attacks are less relevant, because physical access is limited. However, the development of the omnipresent Internet of Things (IoT), or the connected car increases the amount of embedded appliances, which can be attacked using these side-channels. Hence, it is paramount to secure the cryptographic algorithms not only against attacks on the timing side-channels. In this paper, we analyze the stream cipher ChaCha20 [3], [7] and show how the secret key can be completely extracted using EM-based side-channel attacks.

ChaCha20 is implemented in a straightforward manner as described in IETF RFC 7539 [3]. We propose two key extraction attacks. The first attack assumes full control over the message counter and the nonce. Next we propose an advanced attack, which makes it possible to successfully extract the full key even with limited control over the message block counter and the nonce. In particular, an attacker needs only the control over two words of the nonce to first extract the key from two columns of the ChaCha state, followed by an extented attack on the second round. With the full intermediate state, the attacker can apply the inverse permutation of the quarter-round function to extract the full key. We then discuss how this attack applies to the ChaCha20-Poly1305 authenticated encryption scheme, if the generation of the one-time key for Poly1305 [3] uses ChaCha20. Thereby, an attacker might forge messages that are encrypted correctly and have a valid authentication tag.

As a countermeasure, we analyze shuffling of the ChaCha20 ARX operations [8]. Shuffling is a basic countermeasure and often considered as a noise generator to increase the attack effort. This countermeasure may achieve a lower runtime overhead compared to other countermeasures that are provable secure against first-order side-channel attacks, such as threshold implementations [9] which have huge overheads and are hence, shuffling is more suitable for lightweight embedded targets with lower security requirements. By analysis tools like TVLA and NICV in a controlled and simulated setting, we analyze the effectiveness of our proposed shuffling in theory. Results indicate that shuffling the ChaCha20 can reduce the leakage by a factor up to $7\times$.

The remainder of this paper is organized as follows. Sec. II recalls background concepts related to side-channel analysis and shuffling. The ChaCha20 algorithm is presented in Sec. III. Our proposed attacks are described in Sec. IV. The analysis of shuffling as a side-channel countermeasure for ChaCha20 is given in VI. Practical implementation results are reported in Sec. VII followed by final conclusions in Sec. VIII.

II. Related Work

A. *Side-Channel Analysis*

Side-channel analysis (SCA [5]) is considered a critical threat to cryptographic algorithms. SCA exploits unintentional leakage observed in the various physical channels like timing [4], power consumption [5], or electromagnetic (EM) emanation [6]. As discussed before, ARX ciphers are designed to eliminate timing side-channel leaks. Therefore, we focus on power- and EM-based exploits. To exploit the side-channel leakage, one targets a key-related sensitive computation being performed by the target device based on some leakage model assumptions. For microcontrollers, two leakage models are often used. The system bus often leaks with Hamming weight model $HW(.)$, which is the number of set bits (`'1'`) on the system bus. The data registers are assumed to leak with Hamming distance model $HD(.)$, which equals the number of bit transitions, when the value in the register changes from one state to another.

The attack finds a dependency between the observed leakage (or traces) and the hypothetical sensitive values computed using the leakage model and the key hypotheses. An attack is successful when the correct key hypothesis stands out from all other key hypotheses. The dependency is computed by statistical tools like difference of means (DoM [5]), or Pearson correlation coefficient [10]. In the following, we use the Pearson correlation coefficient with the $HW(.)$ leakage model.

B. Leakage Detection

A typical side-channel trace may consist of a million of samples. Processing traces of such size requires a lot of time and computing power. To optimise the attack, it is desirable to focus on a relevant section (few samples) of the traces to accelerate the attack as well as to avoid ghost peaks from irrelevant sections of the trace. This process to find the relevant samples or points of interest (PoI) in a trace is called leakage detection and is a pre-processing step. We use Normalized Inter-Class Variance (NICV [11]) mainly for leakage detection. It is computed as:

$$NICV = \frac{\mathsf{Var}\left[\mathbb{E}\left[Y|X\right]\right]}{\mathsf{Var}\left[Y\right]} \ , \tag{1}$$

where Y denotes a side-channel trace and X is the public parameter (plaintext/ciphertext/nonce), used to partition the traces. $\mathbb{E}\left[.\right]$ and $\mathsf{Var}\left[.\right]$ are statistical expectation and variance. NICV is bounded in the range $[0, 1]$. The key advantage of NICV is that it does not depend on the underlying leakage model but only on public input or outputs. NICV also allows to compute the signal-to-noise (SNR) ratio of the measurements by the following relationship:

$$NICV = \frac{1}{1 + \frac{1}{SNR}} \ . \tag{2}$$

C. Leakage Assessment

To certify or evaluate a cryptographic implementation, a purely attack based methodology can be tedious and non-trivial, because the list of attacks is ever increasing and the evaluation depends highly on the expertise of the attacker. As a counter proposal, conformance-based methodologies are gaining popularity with the Test Vector Leakage Assessment (TVLA [12]) as the forerunner.

TVLA tests for data dependence of the side-channel traces. Several variants of TVLA exist but we focus on the fixed vs random (FVR) test. It partitions traces based on varying and fixed plaintext (or nonce). Next, a hypothesis testing is performed on the two sets. Assuming a null hypothesis that the mean of two sets is identical, Welsh's t-test is performed on the two sets. It is computed as:

$$TVLA = \frac{\mu_r - \mu_f}{\sqrt{\frac{\sigma_r^2}{m_r} + \frac{\sigma_f^2}{m_f}}} \ , \tag{3}$$

where μ_r, σ_r and m_r are mean, standard deviation and cardinality of the set with varying plaintexts. Similarly, μ_f, σ_f and m_f are mean, standard deviation and cardinality of the set with fixed plaintexts. The null hypothesis is accepted only if the TVLA value stays in the range $[-4.5, 4.5]$ with a confidence of 99.9999%. A rejected null hypothesis implies that the device leaks side-channel information.

D. Shuffling

A basic SCA assumes that the order and the timing of operations in an implementation remains the same irrespective of the inputs. Thus an attacker can correlate a specific time sample in the collected set of traces to the hypothetically modelled leakage. Shuffling [8] is a side-channel countermeasure which randomises the order of the execution of sensitive operations in order to reduce the dependency between side-channel traces and the key dependent sensitive variable. Shuffling distributes a particular leakage sample across t time sample, where t is the order of the shuffling. Most shuffling countermeasures permute a set of independent operations (such as S-box lookup). It reduces the absolute correlation (and also NICV) by a factor of $\sqrt{t}$ [8]. However, if the operations are not independent the security gain will be less. In other words, shuffling increases the attack effort and can be considered as a noise source for side-channel protection.

E. Side-Channel Analysis for Salsa

To the best of our knowledge, there is no previous published work on EM or power analysis attacks on ChaCha20. However, some work has been done on Salsa20, which is a close relative of ChaCha20. The authors of [13] conclude that there is a high potential for side-channel attacks exploiting the power side-channel, using either simple or differential power analysis. A successful attack against Salsa20 is presented in [14]. The specification of Salsa20 differs only slightly from ChaCha20. However, the input words are reordered and the ChaCha20 permutation processes the input words in a different way. Therefore, the previous work is not applicable to ChaCha20 and a new analysis is needed.

F. Countermeasures for ARX ciphers

There are only very few side-channel countermeasures for ARX ciphers published in the literature. The two best known candidates are based on masking [15] and its variant threshold implementation (TI) [9]. ChaCha20 uses both linear operations (XOR, rotation) as well as modular addition. Thererfore, a direct application of Boolean masking or TI is costly. A slightly more efficient masking scheme converts the mask values between Boolean and arithmetic masking for the linear operations and the modular addition, respectively. Both approaches have been evaluated in literature for HMAC-SHA-1 and Speck [16], [17], [18]. In contrast to these first- or higher-order secure countermeasures, our goal is to investigate, if shuffling is a low-cost medium-security alternative.

III. The ChaCha20 Algorithm

The ChaCha20 stream cipher was proposed in [7] as a variant of the stream cipher Salsa [19]. Our description of ChaCha20 follows IETF RFC 7539 [3].

ChaCha20 is based on the quarter-round funtion shown in Algorithm 1. This function only contains (modular) additions, XORs and rotations.

The quarter-round function is used to build the double-round function (Algorithm 2), which computes two rounds of ChaCha20. One of these rounds is a so-called *column* round, whereas the other one is a *diagonal* round. This naming convention comes from a two dimensional interpretation of the ChaCha20 state. The double-round function is executed 10 times, i.e. ChaCha20 uses 20 rounds in total.

The initial state of ChaCha20 is set to the following values:

$$\begin{array}{llll} v_0 \leftarrow C_0, & v_1 \leftarrow C_1, & v_2 \leftarrow C_2, & v_3 \leftarrow C_3 \\ v_4 \leftarrow k_0, & v_5 \leftarrow k_1, & v_6 \leftarrow k_2, & v_7 \leftarrow k_3 \\ v_8 \leftarrow k_4, & v_9 \leftarrow k_5, & v_{10} \leftarrow k_6, & v_{11} \leftarrow k_7 \\ v_{12} \leftarrow \text{count}, & v_{13} \leftarrow n_0, & v_{14} \leftarrow n_1, & v_{15} \leftarrow n_2 \end{array},$$

where $C_0, \ldots, C_3$ are pre-defined constants, $k_0, \ldots, k_7$ are the key words split into 8 words, count is a message block counter and n_0, n_1, and n_2 are three words forming the 96 bit nonce.

After the computation of the 20 rounds, the end result (state_{20}) is added to the initial state (state_0), i.e. an output block of ChaCha20 is generated by computing $\text{block} \leftarrow \text{state}_0 + \text{state}_{20}$.

A message is encrypted using ChaCha20 by generating a key stream (one or several blocks) and combining this key stream and the message with XOR. For encrypting messages that are longer than 512 bit, the message block counter is incremented for each message block.

IV. Side-Channel Evaluation

In this section, we propose two key-extraction attacks on an unprotected ChaCha20 implementation. For the evaluation, we implemented ChaCha20 on an Arduino Due platform, which features an Atmel SAM3X8E processor. This processor uses an ARM Cortex-M3, and hence has native support for processing 32 bit addition, rotate and XOR.

Our measurement setup consists of a Langer near-field RF-U 5-2 probe placed directly over the processor die. The traces are captured by using a LeCroy HDO6104-MS oscilloscope with a sampling rate of 2.5 GSamples/s. We use the segmented memory in the scope to accelerate the measurement process. The overall time required for measurement and analysis stays below 30 minutes.

Algorithm 1 The quarter-round function of ChaCha20

```
Require: $a, b, c, d \in \mathbb{Z}_2^{32}$
Ensure: $(a_2, b_4, c_2, d_4) = \text{quarter-round}(a_0, b_0, c_0, d_0)$
  $a_1 \leftarrow a_0 + b_0, \quad d_1 \leftarrow d_0 \oplus a_1, \quad d_2 \leftarrow d_1 \lll 16$
  $c_1 \leftarrow c_0 + d_2, \quad b_1 \leftarrow b_0 \oplus c_1, \quad b_2 \leftarrow b_1 \lll 12$
  $a_2 \leftarrow a_1 + b_2, \quad d_3 \leftarrow d_2 \oplus a_2, \quad d_4 \leftarrow d_3 \lll 8$
  $c_2 \leftarrow c_1 + d_2, \quad b_3 \leftarrow b_2 \oplus c_2, \quad b_4 \leftarrow b_3 \lll 7$
  return $(a_2, b_4, c_2, d_4)$
```

Algorithm 2 The double-round function of ChaCha20

```
Require: $v = (v_0, \ldots, v_{15}), v_0, \ldots, v_{15} \in \mathbb{Z}_2^{32}$
Ensure: $v'' = \text{double-round}(v)$
  $(v'_0, v'_4, v'_8, v'_{12}) \leftarrow \text{quarter-round}(v_0, v_4, v_8, v_{12})$
  $(v'_1, v'_5, v'_9, v'_{13}) \leftarrow \text{quarter-round}(v_1, v_5, v_9, v_{13})$
  $(v'_2, v'_6, v'_{10}, v'_{14}) \leftarrow \text{quarter-round}(v_2, v_6, v_{10}, v_{14})$
  $(\boldsymbol{v'_3}, \boldsymbol{v'_7}, \boldsymbol{v'_{11}}, \boldsymbol{v'_{15}}) \leftarrow \text{quarter-round}(\boldsymbol{v_3}, \boldsymbol{v_7}, \boldsymbol{v_{11}}, \boldsymbol{v_{15}})$

  $(v''_0, v''_5, v''_{10}, \boldsymbol{v''_{15}}) \leftarrow \text{quarter-round}(v'_0, v'_5, v_{10'}, \boldsymbol{v'_{15}})$
  $(v''_1, v''_6, \boldsymbol{v''_{11}}, v''_{12}) \leftarrow \text{quarter-round}(v'_1, v'_6, \boldsymbol{v'_{11}}, v'_{12})$
  $(v''_2, \boldsymbol{v''_7}, v''_8, v''_{13}) \leftarrow \text{quarter-round}(v'_2, \boldsymbol{v'_7}, v'_8, v'_{13})$
  $(\boldsymbol{v''_3}, v''_4, v''_9, v''_{14}) \leftarrow \text{quarter-round}(\boldsymbol{v'_3}, v'_4, v'_9, v'_{14})$
  return $v''$
```

A. CPA on Unprotected ChaCha

For our attack on ChaCha20, we used the principles of correlation power analysis (CPA). The same principles also apply for EM measurements. We use the Pearson correlation coefficient to rank different hypothetical power or EM values by computing the correlation of the hypothetical values with the sample points of the captured power or EM traces [10]. We use the following formula to compute the Pearson correlation coefficient:

$$r = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{\sqrt{(\sum x_i^2 - n\bar{x}^2)}\,\sqrt{(\sum y_i^2 - n\bar{y}^2)}}. \tag{4}$$

As our attacker model, we use the HW model on two different intermediate values. For the first analysis, we assume that the attacker has full control of the message block counter and the nonce. Then, we can attack the key words $k_0, \ldots k_3$ with the following model:

$$HW(d_1) = HW(d_0 \oplus a_1) = HW(d_0 \oplus (a_0 + b_0)), \tag{5}$$

where the input a_0 is one of the constants C_i, b_0 is the key word that we want to attack and d_0 is a word of the nonce or the message block counter under the control of the attacker. For the attack, we could choose to compute hypotheses for all 32 bits of the key word, but this becomes computationally costly to mount. Another approach is divide and conquer, i.e. we extract only a subset of bits of the key at a time and combine the individual results. In the current work, we extract each key word byte by byte.

The first attacked key byte is the least significant byte (LSB) of the key word. For the next key bytes, the results of the previous bytes have to be combined with the key hypothesis for the next byte, because of the modular addition with the constant. For example, if we extracted two equally ranked key hypotheses for the LSB, then we compute in total the correlation coefficient for $2 \cdot 256 = 512$ models for the second key byte.

For the model depicted in Eq. 5, we will always get two hypotheses for b_0 with the same absolute correlation, because of the symmetry of XOR. An attacker which extracted these key hypotheses can easily just try both key hypotheses. However, it is also possible to select the correct hypothesis for the keys $k_0, \ldots k_3$ based on the structure of ChaCha20. In particular, we attack the second half of the key words ($k_4, \ldots k_7$) using

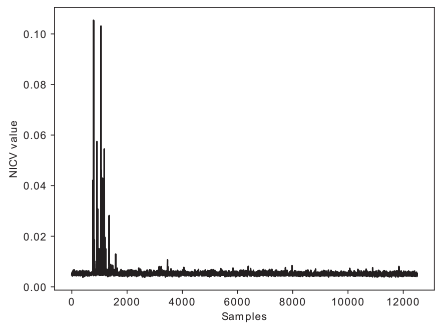

Fig. 1: NICV for the measured traces.

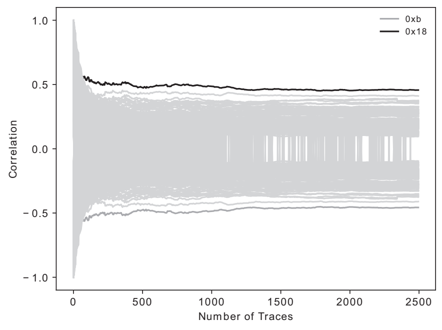

Fig. 2: Correlation vs. number of traces for k_1.

the HW model shown in Eq. 5. Again, the attack is carried out in a divide and conquer fashion, byte by byte.

$$\begin{aligned} HW(b_1) &= HW(b_0 \oplus c_1) = HW(b_0 \oplus (c_0 + d_2)) \quad (6) \\ &= HW(b_0 \oplus (c_0 + (d_1 \lll 16))) \end{aligned}$$

Using this HW model does not only identify the correct key hypotheses for c_0, but in doing so, it also rejects the wrong key hypothesis for b_0, because of the non-linear modular addition and the second usage of b_0.

The pratical attack was carried out in the following order. First a set of 50000 traces was measured using the setup described above. Then we checked with NICV, if our measurements contained any leakage at all to make sure that our measurement setup was working in principle (Fig. 1). NICV also helps in selecting relevant lekage samples, thus leading to an acceleration of attack. Afterwards, we performed the actual CPA on the traces.

The attack results are shown in Fig. 2 and Fig. 3, for the first byte of k_1 and k_5. In these figures, we show the correlation on the y axis and the number of traces on the x axis. As expected, the plot for k_1 is symmetrical and we can see, that after about 100 traces, two hypotheses are standing out with the same absolute correlation. The correct key hypothesis `0x18` is in black and the symmetric counterpart with `0xb` in dark gray. In comparison, for k_5 only one key hypothesis has a distinguishable correlation after roughly 200 traces. In this way, the hypothesis for k_5 also results in a selection of the correct hypothesis for k_1.

B. CPA with Reduced User-Controllable Input

Usually, the attacker does not have full control over all of the 128 bits of publicly known variable input to ChaCha20. In this section, we outline an attack, which makes it possible to extract the full internal state of ChaCha20, as long as the attacker has control over two input words of the nonce per encryption. Furthermore, it is assumed that the other parts of the public input are constant, i.e. the message block counter is fixed to a constant value and the remaining word of the nonce is constant. These are reasonable assumptions, as per RFC 7539, the message block counter is fixed to 0 for the key generation for Poly1305 and each word of the nonce is publicly known.

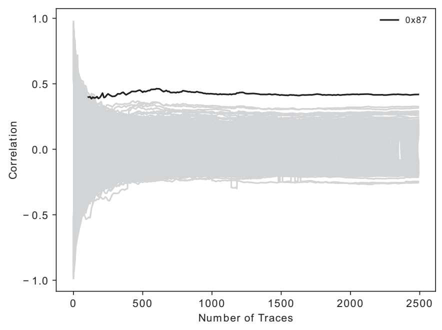

Fig. 3: Correlation vs. number of traces for k_5.

Now, let us assume that an attacker has control over the two input words of the initial state v_{14} and v_{15}. Then he is able to recover 128 bits of the key using the previously proposed attack. In the second round the attacker then controls two input words for each execution of the quarter-round function. With this control, it is possible to successfully recover the remaining parts of the internal state.

For each position, we have to use a slightly different model, because the attacker knows the words $v_2, v_3, v_6, v_7, v_{10}, v_{11}, v_{14}, v_{15}$ and can compute the outputs v' for these two columns (refer to Alg. 2).

- Control of a_0, d_0 (v'_3, v'_{14})
  1) Attack is equal to the first attack.
- Control of a_0, b_0 (v'_2, v'_7)
  1) Extract d_0 with $HW(d_1) = HW(d_0 \oplus (a_0 + b_0))$.
  2) Extract c_0 with $HW(b_1) = HW(b_0 \oplus c_1) = HW(b_0 \oplus (c_0 + (d_1 \lll 16)))$.
- Control of b_0, c_0 (v'_6, v'_{11})
  1) Extract d_2 with $HW(b_1) = HW(b_0 \oplus (c_0 + d_2))$.
  2) Extract a_2 with $HW(d_3) = HW(d_2 \oplus a_2) = HW(d_2 \oplus (a_1 + (b_1 \lll 12)))$.
- Control of c_0, d_0 (v'_{10}, v'_{15})

1) Extract a_1 with $HW(c_1) = HW(c_0 + ((d_0 \oplus a_1) \lll 16))$.
2) Extract b_0 with $HW(b_1) = HW(a_1 + ((b_0 \oplus c_1) \lll 12))$.

V. Impact on Poly1305

The Poly1305 algorithm itself uses one-time keys for every generation of a MAC [20]. Hence, it is not directly exploitable by a correlation EM analysis. However, the key generation can be attacked, if Poly1305 is used in combination with ChaCha20, because ChaCha20 is used to generate this key according to RFC 7539 [3].

In particular, the key for Poly1305 is generated by one call to the key stream generation of ChaCha20, with the encryption key, a user-defined nonce and the message block counter set to zero. The public nonce itself is generated by concatenating a user-defined initialization vector and optionally by a constant. If the initialization vector (IV) is less than 96 bits in length, it is prepended by a constant which is of length $96 - |\text{IV}|$.

According to our results in Sec. IV-B, we can mount a side-channel attack on the second quarter round of ChaCha20 using a power or EM correlation analysis, if the attacker has full control over at least two 32 bit input words. Hence, a side-channel attack on the key generation of Poly1305 is possible, breaking the authentication in ChaCha20-Poly1305. Furthermore, since the same key is used in the encryption with ChaCha20 not only the authentication is compromised, but also the encryption.

VI. The Shuffling Countermeasure

In this section, we analyze the possibility to use shuffling as a low-cost hiding countermeasure.

The first possibilty to shuffle operations of the ChaCha20 algorithm is to shuffle the different executions of the quarter-round function. However, this only leads to $4! = 24$ possible different combinations, which is not very useful as a protection mechanism. A better option is to shuffle all operations of a full round, of course with the condition that the algorithm result is still correct. This leads to $\frac{(nm)!}{(m!)^n}$ possible orderings, where n is the number of independent sequences of operations, and m is the number of operations per sequence. For the ChaCha20 round fuction, the parameters are $n = 4$ and $m \leq 12$, because we have four independent quarter-round functions with 12 operations each.

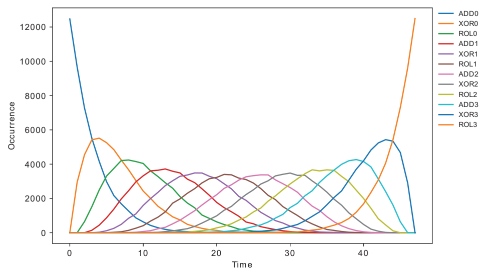

Fig. 4: Distribution of different parts of a quarter-round

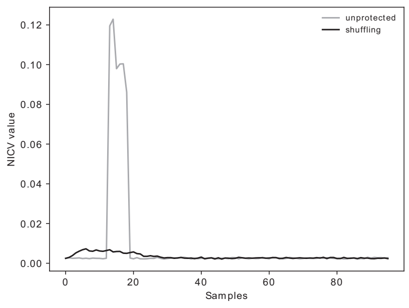

Fig. 5: NICV for shuffling using simulated traces and noise with standard deviation 3

However, the individual operations are not independent of each other. Thus, shuffling affects some operations more than others leading to a bias. In particular, the distribution of the operations follows Fig. 4, if the shuffling is done for all operations of one round function. For this figure, we simulated 50000 different permutations and plotted the occurence of the the individual operations of one quarter-round function. We can see, that the first and the last operation have a high likelihood to be executed in the beginning or the end respectively, while the peaks of the distribution of the other operations are much lower. This is a favorable property, since we usually do not attack the first and last operations of the ChaCha20 quarter-round.

An analysis of the shuffling countermeasure with simulated traces shows a similar behavior, when analysed with NICV. Fig. 5 shows the NICV for the first byte of the nonce n_1, which relates with the processing of first byte of key k_1. Since the computation for the unprotected implementation is synchronised, we see a sharp and high peak corresponding to this particular byte. When the same analysis is performed on the shuffled implementation, the leakage is much lower and spread over time. The spread over time indicates that a particular operation can be executed on different points of time, based on the permutation, but still statistically observable on side-channel.

Next the leakage reduction due to shuffling is analysed using TVLA. We compared the TVLA values of the simulated traces of the unprotected version with our shuffling countermeasure, under different noise settings. As shown in Fig. 6, both unprotected and shuffled ChaCha20 leak sensitive information, much above the threshold of $|4.5|$. However, the inclusion of the shuffling countermeasure does reduce the leakage by a huge factor. The leakage detection reduction (shown as shaded area in Fig. 6) is much higher in the noise standard deviation range $[0, 3]$, which is closer to relaistic scenarios. At its peak, the detected leakage is reduced by up to $7\times$. As expected, shuffling does not remove the side-channel leakage but only make it harder to detect and exploit by an attack.

VII. Performance Evaluation

The main reason for evaluating the shuffling countermeasure was to check its potential as a suitable lightweight counter-

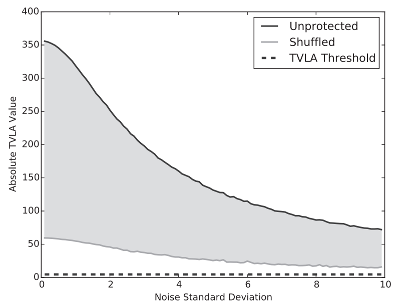

Fig. 6: TVLA Leakage for unprotected and shuffled ChaCha20 under varying noise

measure. This might be interesting for applications, which cannot afford resource consuming provable first- or higher-order countermeasures. The implementation of our countermeasure does not compare very favorable to the unprotected version, as it adds three `LOAD`, one `STORE`, two `ADD` and two `BRANCH` instructions to the program flow for each operation. Therefore, the runtime per round increases from **105** clock cycles to **1927** clock cycles, which is an almost $20\times$ overhead. Additionally, each shuffling permutation has to be generated, which takes some additional time. Thus, shuffling at considerable order does not turn out to be lightweight.

The main reason for this overhead is that it is difficult to randomize the operations in the proposed way efficiently. Currently, we use one array to keep track of the overall computation. This array directs the program flow to advance one of the four quarter-round functions at a time. The second array keeps track of each quarter-round function individually, i.e. it knows the state of each quarter-round function. The third array points to the individual operations and the entries can be used as branch targets. Using only one lookup table which directly points to the respective code portion is possible. However, this will be paid for with an increased overhead when generating the permutation, due to the dependencies between the different operations of the quarter-round.

VIII. Conclusion

ChaCha20 is a standardized stream cipher based on an ARX construction. Due to this construction, it is widely believed to be secure against side-channel analysis. However, this is only true for timing-based side-channel. In this paper, we investigate EM-based side-channel attacks on ChaCha20. Two attacks are presented, one assuming a strong attacker with full control over the nonce while the other assumes a weak attacker having only partial control. The proposed attack is also applicable on Poly1305 based key generation which itself uses ChaCha20. Finally the potential of using a shuffling countermeasure to mitigate side-channel leakage is explored. Although shuffling can significantly reduce the SNR of the leakage, the countermeasure still remains vulnerable and can be potentially exploited with added effort. Future works could explore alternative lightweight countermeasures for ARX constructions.

Acknowledgement

The authors would like to thank Dr. Marc Stöttinger, Independent researcher, and Richard Petri from Fraunhofer SIT for their kind help with the measurement of the side-channel traces and additional technical comments that greatly improved the manuscript.

References

[1] E. Bursztein, "Speeding up and strengthening HTTPS connections for Chrome on Android," 2014, https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html.
[2] M. Staruch, "Support for ChaCha20-Poly1305," 2015, https://github.com/openssl/openssl/issues/304.
[3] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," IETF RFC 7539, 2015.
[4] D. J. Bernstein, "Cache-timing attacks on AES," 2005, https://cr.yp.to/antiforgery/cachetiming-20050414.pdf.
[5] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.
[6] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM Side—Channel(s)," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2002, pp. 29–45.
[7] D. J. Bernstein, "ChaCha, a variant of Salsa20," in *Workshop Record of SASC - The State of the Art of Stream Ciphers*, 2008.
[8] N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert, "Shuffling against side-channel attacks: A comprehensive study with cautionary note," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2012, pp. 740–757.
[9] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *International Conference on Information and Communications Security*. Springer, 2006, pp. 529–545.
[10] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2004, pp. 16–29.
[11] S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, "Side-channel leakage and trace compression using normalized inter-class variance," in *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2014, p. 7.
[12] B. J. Gilbert Goodwill, J. Jaffe, P. Rohatgi *et al.*, "A testing methodology for side-channel resistance validation," in *NIST non-invasive attack testing workshop*, 2011.
[13] B. Gierlichs, L. Batina, C. Clavier, T. Eisenbarth, A. Gouget, H. Handschuh, T. Kasper, K. Lemke-Rust, S. Mangard, A. Moradi, and E. Oswald, "Susceptibility of eSTREAM candidates towards side channel analysis," in *Workshop Record of SASC - The State of the Art of Stream Ciphers*, 2008.
[14] B. Mazumdar, S. S. Ali, and O. Sinanoglu, "Power analysis attacks on ARX: An application to Salsa20," in *2015 IEEE 21st International On-Line Testing Symposium (IOLTS)*. IEEE, 2015, pp. 40–43.
[15] J.-S. Coron and L. Goubin, "On Boolean and arithmetic masking against differential power analysis," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2000, pp. 231–237.
[16] J.-S. Coron, J. Großschädl, and P. K. Vadnala, "Secure conversion between Boolean and arithmetic masking of any order," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2014, pp. 188–205.
[17] J.-S. Coron, J. Großschädl, M. Tibouchi, and P. K. Vadnala, "Conversion from arithmetic to Boolean masking with logarithmic complexity," in *International Workshop on Fast Software Encryption*. Springer, 2015, pp. 130–149.
[18] T. Schneider, A. Moradi, and T. Güneysu, "Arithmetic addition over Boolean masking," in *International Conference on Applied Cryptography and Network Security*. Springer, 2015, pp. 559–578.
[19] D. J. Bernstein, "The Salsa20 family of stream ciphers," in *New stream cipher designs*. Springer, 2008, pp. 84–97.
[20] ——, "The Poly1305-AES message-authentication code," in *International Workshop on Fast Software Encryption*. Springer, 2005, pp. 32–49.